sockgw - a stream socket to UDP gateway

Srini, rs@toprllc.com

July 10, 2021

1 Introduction

This projectlet is an answer to a real issue found in a device.

Background The user interface layer of the device referred to earlier was a **Qt** based application that interacted with multiple touchpanel displays. The application received data from the lower level controllers through a stream socket. Requirements specified creating logs of the data streams received in addition to extracting and displaying specific data items.

Challenge and Opportunity The single application having to deal with multiple devices, data streams and creation of files led to a fairly complex design leading to multiple threads, complex interactions and thereby a system that is hard to diagnose and make incremental improvements to.

The solution A potential solution was proposed and this projectlet is a **proof of concept** implementation. The stream socket is handled by an independent process that receives the data, performs the logging function and forwards selected data records to the GUI application. With the data logging removed, the GUI subsystem became considerably simpler and was able to focus on the user interface leading to a simpler internal design.

1.1 Learning Objectives

Stream Socket server Stream socket servers create a **listening** socket where the servers can wait for connection requests. Every connection request is closed using an **accept socket** which can now be independently serviced. This allows a server to provide the service to many clients simultaneously. The stream sockets however view the data as a stream of data and any message/record delineations are left to the application. In this example, we assume that each message is terminated with an **end of line** indicator. Other patterns may utilize for example the exact same size of message or a message header that indicates the size of the message.

Datagram sockets Datagram sockets may be simplistically understood as **best effort** message transfer mechanism. In other words, messages may get lost or arrive out of sequence and thus using datagram sockets in an application requires careful and deliberate design. On the other hand, this approach may have relatively low overhead and complexity since it

is easy to maintain record boundaries. In this application, the assumption of being on the same node led to this mode for message transfer to the GUI application.

Design for testing The projectlet supports a variety of command line options leading to the ability to test the core features without requiring the actual application. A test traffic generator, a receiver for the converted datagrams are additional modes supported in addition to the gateway mode.

2 Specifications

```
$ ./main -h
sockgw V01 2021-04-22 05:20:24
Usage: main.exe sockgw

 -v, --verbose           Output extra verbose information
 -L, --logging-level=ARG Logging Level
 -c, --client            Start sample client
 -u, --udptarget         Start a UDP target server
 -g, --gateway[ARG]      Gateway Stream Socket Port Number
 -t, --target[ARG]       Target UDP Socket Port Number
```

2.1 Examples

Gateway mode In this mode the application just waits for client connections and each stream is forwarded to the UDP server in addition to creating logs.

```
 $ ./main -g
Received a connection Client: 127.0.0.1:49849
Server No  1
 1> Forwarded  25 bytes
 1> Forwarded  25 bytes
 1> Forwarded  25 bytes
 1> Forwarded  25 bytes
 1> Forwarded  25 bytes
 1> Forwarded  25 bytes
 1> Forwarded  25 bytes
 1> Forwarded  25 bytes
 1> Forwarded  25 bytes
Received a connection Client: 127.0.0.1:49853
Server No  2
 2> Forwarded  25 bytes
 2> Forwarded  25 bytes
 2> Forwarded  25 bytes
 2> Forwarded  25 bytes
Received a connection Client: 127.0.0.1:49865
Server No  3
 3> Forwarded  25 bytes
```

```
 3> Forwarded  25 bytes
 3> Forwarded  25 bytes
```

The log files While forwarding the packets, the data is also output to a log file:

```
$ ls -l archive_20210*.bin
-rw-r--r-- 1 RajaS 197609    0 Apr 22 05:05 archive_20210422_050520.bin
-rw-r--r-- 1 RajaS 197609 3550 Apr 22 05:12 archive_20210422_050958.bin
-rw-r--r-- 1 RajaS 197609 2250 Apr 22 05:40 archive_20210422_053814.bin
-rw-r--r-- 1 RajaS 197609    0 Jul 10 04:43 archive_20210710_044352.bin
```

Target mode In this (target UDP server) mode, the data received is simply printed out. As this trace shows, the server is receiving several independent streams of messages. Looking at the timestamps, 3 clients started close to each other, sending a packet every 5 seconds and a 4th client also sending a packet every 5 seconds except offset by 1 second.

```
 $ ./main -v -u -L=100
2021-07-10 07:20:03.00 [INFORMATIONAL] This is message 3
2021-07-10 07:20:03.00 [INFORMATIONAL] This is message 31
2021-07-10 07:20:07.00 [INFORMATIONAL] This is message 34
2021-07-10 07:20:08.00 [INFORMATIONAL] This is message 15
2021-07-10 07:20:08.00 [INFORMATIONAL] This is message 4
2021-07-10 07:20:08.00 [INFORMATIONAL] This is message 32
2021-07-10 07:20:12.00 [INFORMATIONAL] This is message 35
2021-07-10 07:20:13.00 [INFORMATIONAL] This is message 16
2021-07-10 07:20:13.00 [INFORMATIONAL] This is message 5
2021-07-10 07:20:13.00 [INFORMATIONAL] This is message 33
```

Test Traffic mode In this mode the application connects to the socket gateway and generates messages periodically. The above log came from 4 independent clients invoked as shown below.

```
$ obj/main -c
Connected to        1025127.0.0.1
Sent > No:           1          18 bytes
Sent > No:           2          18 bytes
Sent > No:           3          18 bytes
Sent > No:           4          18 bytes
Sent > No:           5          18 bytes
Sent > No:           6          18 bytes
Sent > No:           7          18 bytes
Sent > No:           8          18 bytes
Sent > No:           9          18 bytes
```

3 Implementation

Implementation in **Ada** `https://gitlab.com/ada23/sockgw.git`